

**REMARKS**

[0001] The Office Action rejected Claims 1-22 under 35 U.S.C. §103(a) as being unpatentable over Sand, et al (US Patent No. 6,148,322, hereinafter Sand) in view of Reznak (U.S. Pat. No. 6,601,083, hereinafter Reznak).

[0002] The Applicants wish to traverse the rejection of Claims 1-22 under 35 U.S.C. §103(a). For the reasons set forth below, the Applicants respectfully request that the rejections be withdrawn and that the Claims 1-22 be allowed.

**REJECTION OF CLAIMS 1-22 UNDER 35 U.S.C. §103(a)**

[0003] The Office Action rejected independent Claims 1, 8, 9, 16 and dependent Claims 2-7, 10-15, 17-22 under 35 USC §103(a) in view of Sand and Reznak.

[0004] Sand and Reznak both teach methods of controlling high priority computer tasks using a preemptive model. In a preemptive multitasking environment, the operating system allows each task to run for a limited time-slice. When the time slice for a given task ends, the operating system suspends the task and allows another task to run. In a non-preemptive environment, each task runs until the task voluntarily gives up control.

[0005] **Sand**. Sand generally teaches a means to prevent a high priority task from completely shutting out a low-priority task using a preemptive enforcement model. Specifically, Sand teaches that each task request a time-slice by specifying a request condition and a termination condition. However, a high priority task will be aborted if the high priority task exceeds its time-slice. "Processing of the higher-priority task continues until a predetermined maximum processing time elapses, at which time the task is

aborted in favor of a task having a next highest priority..." Sand, col. 2, ll. 16-18, (emphasis added). Sand teaches a preemptive multitasking model.

[0006] **Reznak**. Reznak teaches a preemptive method of penalizing tasks which request too many system resources. "[M]emory allocation requests that exceed the task penalty threshold or penalty ratio are not denied, but are serviced after a penalty has been assessed and served by [the] requesting task." Reznak, col. 5, ll. 40-43. The penalty is imposed as an immediate preemptive suspension of the offending task. Reznak, col. 6, ll. 36-38; Figure 5, blocks 122, 124; Figure 6, blocks 152, 160, 168. Reznak teaches that the managed system resource is system memory. Reznak does not teach that time slices themselves are a system resource. In fact, Reznak teaches that the loss of time slices is to be used as a penalty for requesting too many system resources. Reznak, col. 6, ll. 36-38.

[0007] To establish a *prima facie* case of obviousness, the combination of prior art references must teach or suggest all the claim limitations. MPEP §2142. In addition, "it is insufficient that the prior art disclosed the components of the patented device, either separately or used in other combinations; there must be some teaching, suggestion, or incentive to make the combination made by the inventor." *Northern Telecom, Inc. v. Datapoint Corp.*, 908 F.2d 931, 934 (Fed. Cir. 1990).

[0008] Applicants respectfully assert that Sand and Reznak fail to teach or disclose each element and limitation recited in independent Claim 1. Namely, Sand and Reznak fail to teach

- 1) a non-preemptive "method for self-throttling the use of computer resources by a computer task," wherein a computer task "receiv[es] ... a throttle

specification” and “execute[s] ... until a first unit of work is completed”

- 2) a “method for self-throttling,” directed by the operating system wherein the “computer system” sends a “throttle specification” to a “computer task” which is “recciv[ed] by said computer task;” and used by the computer task “for directing said computer task’s usage of said computer resources”
- 3) prior use of system resources affects the rationing of future use of the same system resource wherein the “suspension time for said computer task [is] based at least partially on said throttle specification and said clapsed time.”

[0009] Neither Sand nor Reznak teaches a non-preemptive, self-throttling method of controlling computer resources; neither Sand nor Reznak teaches a method for self-throttling wherein the throttled task receives instructions from the operating system and directs its use of computer resources according to the directions from the operating system. Although Reznak teaches a method of penalizing tasks which request an excessive amount of system resources, Reznak does not teach a method of rationing future use of a resource based on prior use of the same resource. In short, three limitations found in the present invention are absent from Sand and Reznak.

[0010] The Office Action asserts that Sand teaches “a mcthod of self-throttling the use of computer resources by a computer task.” Under a system of “self-throttling,” the throttling must be carried out by the entity that is being throttled. Although the references teach means to throttle a computer task, none of them teach a self-throttling means or a means whereby the computer task throttles itself. In the references, the system suspends the computer task; therefore the computer task is not throttling itself.

[0011] Other evidence in the body of Claim 1 supports the position that the method of self-throttling must be carried out by the computer task. The claim limitation "receiving by said computer task a throttle specification for directing said computer task's usage of said computer resources" indicates that the computer task itself will be notified of computer resource limitations. This notification is reasonable only if this is a "method for self-throttling" of computer resources by the computer task itself.

[0012] The Office Action asserts that assignment of a request condition and a terminating condition to each task and the tracking of the elapsed execution time of each task taught in Sand, *see* col. 4, ll. 18-21, 43-47, is equivalent to "receiving...a throttle specification" by the computer task, as recited in Claim 1. Applicants respectfully disagree.

[0013] In Sand, the request condition and the terminating condition are not communicated to the task. Sand does not offer a definition for a request condition, but explains how request conditions are satisfied, *see* col. 1, ll. 41-44, stating "multi-tasking in a processing unit involves assigning 'time slices' to each task for which the request condition is satisfied, so that each task is executed periodically during its allocated time slice." From this, "request condition" is not a new term, but simply the condition that is satisfied before a task in a multi-tasking environment is assigned a time slice, or assigned to the run queue.

[0014] Those familiar with the art will realize that a task in a multi-tasking environment may be assigned to a run queue for various reasons including the arrival of a message directed to the task or the expiration of a timer. These and all other request conditions are received by the operating system, causing the operating system to add the task to the run queue. Tasks on the run queue are executed by the operating system

[0015] In Sand, a computer task does not receive a request condition or a terminating condition. See Sand, col. 2, ll. 13-18. In column 2 starting at line 13, Sand teaches that the request condition for a high priority task is fulfilled while the high priority task is not running. Since the request condition is fulfilled while the task is suspended, the request condition cannot be received by the task. The high priority task does not receive a request condition. Rather, the high priority task is reinstated as a result of the fulfillment of a request condition.

[0016] Under Sand, a task is terminated once its termination condition is satisfied. Sand, col. 4, ll. 25-29. Sand teaches an improved "multi-tasking" system. Sand, col. 2, ll. 8-15. The termination condition is simply the condition which causes the task to be suspended. Sand, col. 2, ll. 13-18. Under most multi-tasking systems, a task terminates if the task completes processing of assigned work or if the task uses up its time slice. Under Sand, a task is terminated if it executes longer than a user-defined maximum processing time. Sand, col. 4, ll. 41-47. Termination of a process for running too long is not equivalent to the reception of a throttle specification that will direct a task's use of a computer resource.

[0017] The fulfillment of a request condition as defined in Sand cannot be equivalent to "receiving a throttle specification." Claim 1 states that a throttle specification is "receiv[ed] by the computer task...for directing said task's usage" of computer resources. In the present invention, the received throttle specification itself contains instructions that direct the use of computer resources. A request condition and a terminating condition are not received by the computer task and are not the equivalent of a "throttle specification" or any type of information that directs the "usage of ... computer resources." Rather, the request

condition was fulfilled prior to execution of the task. Consequently, the operating system provides a time slice to the high priority task. The high priority task is ended when the terminating condition is fulfilled.

[0018] Conditions necessary to start and terminate a task are not equivalent to a throttle specification. Throttle specification was intentionally made an element in Claim 1. The definition of throttle specification is clearly given in the specification. Hence, the meaning chosen by the inventor must govern. See *CCS Fitness, Inc. v. Brunswick Corp.*, 288 F.3d 1359, 1366 (Fed. Cir. 2002); *Phillips v. AWH Corp.*, 03-1268, 1269 (Fed. Cir. 2005).

[0019] Also Sand, col. 2, ll. 13-18, teaches that the system aborts a lower priority task in favor of a higher priority task. The suspension of the lower-priority task is preemptive in nature which is typical of the teachings of Sand and Reznak. In the same section, Sand teaches that once the high priority task has run to the end of its time slice, it too is preemptively aborted. Sand, col. 2, ll. 13-18. Sand teaches a method for the system to throttle computer tasks, not a method of self-throttling. No throttle specification is sent to the high priority task in Sand or Reznak, while the present invention claims that a throttle specification is received by the computer task

[0020] The request condition is simply the arrival of an interrupt or the expiration of a timer which causes a task to be eligible for execution. A request condition is not sent to the eligible task. A throttle specification, however, is clearly "receiv[ed] by [the] computer task for directing...usage of ... computer resources," see Claim 1. The throttle specification is sent to the task and directs the computer task to use a set level of computer resources. The

request condition is not sent to the computer task and has no effect on the computer task's use of computer resources.

[0021] "[E]xecuting said computer task until a first unit of work is completed read in light of the phrase "a method for self-throttling" clearly teaches a non-preemptive determination of execution. The task in Claim 1 determines when the first unit of work is complete.

[0022] The Office Action states that this limitation is equivalent to a cyclically-repeating, low priority task which periodically is aborted in favor of an even lower priority communications task which carries out communications requests made by a high priority task. Sand, col. 2, ll. 47-52. Such an interpretation suggests that the cyclically-repeating task is playing the part of a timer which periodically aborts itself in an effort to allow a lower priority task to run. Applicants respectfully disagree with an interpretation that equates a timer task in Sand with the self-throttling mechanism claimed by the phrase "executing ... until a first unit of work is complete."

[0023] The next claim limitation in Claim 1 clearly demonstrates that the interpretation set forth in the Office Action is not reasonable. Claim 1 continues: "calculating the elapsed time of said first unit of work." If a cyclically repeating task were used by the system to suspend and reinstate a particular task, then there would be no need to measure the elapsed execution time. The cycle of the cyclically repeating task would allow for preemption. However, the present invention is for a non-preemptive, self-throttling method for using computer resources. As such, the computer task itself determines the length

of the first unit of work. The elapsed time for that unit of work is determined such that a suspension time can be calculated.

[0024] This is the only reasonable interpretation of the claim language. Any other interpretation would make the calculation of the elapsed time superfluous. This is further supported by the specification which explains that "the high-priority task invokes an operating system service to note the system clock time at the beginning of a first or next unit of work." (Taken from the published version of the present application, US 2003/0088605, hereinafter Beghtel) Beghtel, ¶25. "Upon completion of this unit of work" the operating system service is called again by the high priority task. Beghtel, ¶25. Thus, "Calculating the elapsed time of said first unit of work" is executed by the task itself in a "self-throttling" method. Any other interpretation would contradict the Claim language.

[0025] The Office Action also asserts that the self-throttling method of the present invention comprising "suspending said computer task for said calculated suspension time prior to execution of said computer task" is equivalent to the preemptive management of a computer task taught by Sand, col. 4, ll. 52-54. This interpretation is not reasonable in light of the earlier arguments explaining that the throttling of the task is carried out by the task itself. The specification further supports this interpretation stating, "the throttle specification may be a binary indicator directing high priority task 210 to either suspend itself or not." Beghtel, ¶ 26. The specification confirms that the throttling is truly a self-throttling method as stated explicitly and implicitly in Claim 1 of the present invention.



[0026] Dependent Claim 7 recites

7. The method of Claim 1 wherein said computer task self-throttles the usage of said computer resources by said computer task in accordance with said throttle specification.

[0027] Dependent Claim 7 explicitly states that the "computer task self-throttles the usage of ... computer resources" in accordance with the arguments made above. This language is supported by the specification. Beghtel, ¶23, 25, 26. Nothing in either Sand or Reznak suggests or teaches a self-throttling means of computer resource management. Instead, Sand and Reznak teach an operating system that controls resource usage through preemption. In addition, self-throttling is not an obvious means of resource management and is not taught by either of the combined references. No motivation to combine exists since the individual elements do not exist in either reference.

[0028] The second limitation found in Claim 1 and lacking from the cited references is a method of self-throttling wherein a) the "computer system" sends a "throttle specification" to a "computer task" which is "receiv[ed] by said computer task" and used by the computer task "for directing said computer task's usage of said computer resources." Under this limitation, the computer task voluntarily follows a directive, the throttle specification, concerning the use of a computer resource.

[0029] References for this limitation have been explained above. The present invention teaches a method which relies on a cooperative effort between the operating system and the executing high priority task. The high priority task must "receive a throttle specification," and the high priority task must use the received "throttle specification for

directing said computer task's usage of said computer resources." Sand does not require a high priority task to cooperate with the operating system or to follow specifications distributed by the operating system. Rather Sand teaches a method wherein the system "ensures the highest-priority task does not monopolize system resources by tracking the elapsed execution time and terminating the highest-priority task if this elapsed time exceeds a predetermined maximum..." Sand Abstract, last sentence. Under Sand, the operating system does not work cooperatively with the tasks running on the system.

[0030] Similarly, Reznak teaches the use of a penalty system imposed by the operating system for tasks which request high levels of memory. Reznak teaches a method of control that relies on no cooperation from the running tasks.

[0031] Finally, Claim 1 of the present invention indicates that a task's use of the system resources during the first unit of work will affect the rationing of that same resource during future periods of work. This is embodied in the language of Claim 1: "calculating a suspension time for said computer task [is] based at least partially on said throttle specification and said elapsed time." The elapsed time comprises time to execute unit of work one. The calculated suspension time prior to commencing work on unit of work two is dependent on the amount of time used to complete unit of work one.

[0032] Under Sand, the penalty imposed for attempting to continue to use the CPU beyond a "maximum processing time" is two-fold: First, the task is preemptively suspended, *see* Sand, col. 4, ll. 43-46; and Second, "[t]he execution of the high-priority task hpT will not be resumed until a user-selected deactivation time elapses." Sand, col. 4, ll. 52-54. It would be unreasonable to equate imposing a user-defined suspension period (the user-

selected deactivation time) with a suspension period based on the length of time that a process previously executed. Under the present invention, the suspension of the high priority task is self-imposed and the suspension time is dependent on the amount of time that the high priority task actually executed during a previous unit of work. Under Sand, the penalty is preemptively imposed and is not dependent on the amount of time actually consumed by the high priority task.

[0033] The penalty imposed by Reznak is dependent on the amount of memory requested. Under Reznak, the memory requested is always delivered, but then a preemptive suspension of the task is immediately imposed after delivering the memory. The period of suspension is dependent on the memory requested but is imposed before the task can actually use the memory, while in the present invention, the resource is completely consumed, and a benefit from that resource has already been derived before any penalty is calculated. Under Reznak, the penalty for over-use of one computer resource (memory) is meted out by rationing another computer resource (time). Under the present invention, over-use of time is self-throttled by adjusting the amount of time that the high-priority task will be suspended.

[0034] To establish a *prima facie* case of obviousness, the combination of the prior art references must teach or suggest all the claim limitations. MPEP §2142. As explained above, the non-trivial limitations found in Claim 1 of the present invention are missing from the cited references. If even one limitation is missing, then a §103(a) obviousness rejection is improper. Applicants respectfully submit that independent Claim 1 is patentably distinct and nonobvious over Sand and Reznak. Furthermore, since neither Sand nor Reznak teach or disclose non-preemptive self-throttling, self-throttling based on a

throttle specification received from the operating system, and a rationing of a computer resource based on prior over-use of the same computer resource, there is no motivation to combine Sand and Reznak.

[0035] To establish *prima facie* obviousness, there must be some suggestion or motivation to modify the reference or to combine reference teachings to arrive at the claimed invention. "The teaching or suggestion to make the claimed combination ... must be found in the prior art, not in applicant's disclosure." MPEP 2143, citing *In re Vaeck*, 947 F.2d 488 (Fed. Cir. 1991). "The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination." See MPEP 2143.01 (emphasis in original), citing *In re Mills*, 916 F.2d 680 (Fed. Cir. 1990).

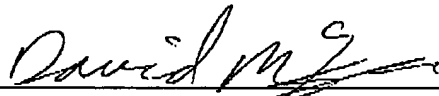
[0036] As discussed above, Sand teaches a method for preemptively suspending a high priority task for a user-defined deactivation time whenever the task "fails to release" the processor for use by others for a specified period of time, and Reznak teaches a method of preemptively suspending a task which has requested an excessive amount of memory before the process can use the memory and for a period of time dependent on the amount of memory requested. However, neither reference teaches or discloses directing a computer task to carry out steps to self-throttle the computer task's use of system resources according to the directions of the operating system or in a way that the future rationing of computer resources will be affected by the prior use of the same resources. Reznak does not suggest any means for the rationing CPU time, let alone the rationing of any resource based on prior use of the same resource. In view of this fact, no motivation can be found to combine Sand and Reznak

for a method of self-throttling the use of computer resources by a computer task. See Claim 1.

[0037] The Office Action rejected independent Claims 8, 9, and 16 under 35 USC §103(a) in view of Sand and Reznak. However, Applicants respectfully submit that Claims 8, 9, and 16 cover substantially the same subject matter as Claim 1 discussed above. Therefore, Claims 8, 9, and 16 should be allowed for at least the same reasons as Claim 1. Furthermore, since Claims 2-7, 10-15, and 17-22, which were also rejected under 35 USC §103(a) in view of Sand and Reznak, depend from Claims 1, 8, 9, and 16 respectively, the Applicants respectfully submit that Claims 2-7, 10-15, and 17-22 are non-obvious for at least the same reasons as Claim 1.

[0038] In view of the foregoing, Applicants submit that the application is in condition for allowance. In the event any questions or issues remain that can be resolved with a phone call, Applicants respectfully request that the Examiner initiate a telephone conference with the undersigned.

Respectfully submitted,



David J. McKenzie  
Reg. No. 46,919  
Attorney for Applicant

Date: August 29, 2005  
8 East Broadway Suite 600  
Salt Lake City, UT 84101  
Telephone (801) 994-4646  
Fax (801) 322-1054